FAULT AWARE LOAD BALANCING AND LEARNING BASED RESOURCE ALLOCATION IN CLOUD

Synopsis submitted in fulfillment of the requirements for the Degree of

Doctor of Philosophy

By

PUNIT GUPTA

Enrollment No. 136209

Under the supervision of

Prof. S. P. Ghrera

Department of Computer Science and Engineering

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT

ABSTRACT

Resource allocation and load balancing in a cloud environment is a problem of interest in recent years. With the increase in number of request over the data centers and size of cloud infrastructure over time, increasing the load unbalancing and power consumption of the data center. So, the requests need to be balanced in such manner having a more effective strategy for resources utilization, request failure, and improved power consumption. Cloud computing made it more complicated with respective to requests types that affect the performance of system.

In general, resource allocation and load balancing algorithm chooses an objective function to select a host with least resource utilization, power consumption to optimize the system performance and provide high Quality of Service.

The objective of this thesis is to bridge the gap between these research directives, 1) load balancing of requests in faulty cloud environment to improve average resource utilization, reduce request failure count, request delay and unbalancing of load among servers, 2) resource allocation and scheduling of requests in faulty cloud where the fault can be due to hardware of software and to improve the performance of system by reducing failure count, failure probability and improve system reliability, 3) improve power efficiency of system by fault and power aware scheduling algorithm for cloud infrastructure.

In the second chapter, set of approaches for request load balancing in a faulty cloud environment to effectively utilizes the server capabilities and resources are proposed. We have also studied and analyzed the algorithm over high request load over a system with various test suites and various request types from the dataset. The results obtained with our approach are compared and analyzed with state of art existing algorithm from literature to study the improvement in performance. Experiment and result shows improvement in average resource utilization and reduces request failure over the system improving QoS over the system by reducing request delay.

In the third chapter, learning based resource allocation algorithm for the faulty cloud which aims to improve the scheduling delay, failure count, failure probability and the reliability of the system are studied and proposed. Propose approaches are not static or dynamic algorithm these are learning based algorithm which aims to learn from past behavior of the system to find a global best solution to have least makespan i.e. total execution time. These algorithms are derived from environment and nature around us. The algorithms are being

studied and analyzed over high request load and variable resources i.e. VM's (virtual machines) to study the behavior of the algorithm in various environments. The Experiments are performed and results are compared with results of various existing proposed algorithms to study the enhancement. The result shows the proposed approached provided least scheduling time, reduced request failure, failure probability, and improved system reliability.

In the fourth chapter, an approach to improve the power consumption and request failure count over the faulty cloud environment is proposed. The algorithm is being tested with the best power efficient existing algorithms like DVFS under various requests load to study the behavior and performance of the algorithm. We obtained improved results of proposed algorithm with lease power consumption and request failure count under various test cases.

Table of Contents

	PAGE NO.
ABSTRACT	ii-iii
1. **INTRODUCTION**	1 - 4
1.1 Load Balancing	1
1.2 Resource Allocation	2
1.3 Popular Algorithms	3
1.4 Parameters Used	4
2. **Fault and Deadline Aware Load Balancing**	5 – 13
2.1 Fault and Load Aware Load Balancing in Cloud Storage	5
2.2 Deadline Aware Load Balancing of Distributed Servers in Distributed	8
2.3 Load Balancing Algorithm for Hybrid Cloud IaaS	10
2.4 Summary and Discussion	13
3. **Learning Based Fault Ware Resource Allocation**	14 – 24
3.1. Fault and QoS based Genetic Algorithm for Task Allocation in Cloud Infrastructure	14
3.2. Task Allocation Using Big Bang-Big Crunch in Cloud Infrastructure	13
3.3. Fault Tolerant Big Bang-Big Crunch for Task Allocation in Cloud Infrastructure	18
3.4. Load and Fault Aware Honey Bee Scheduling Algorithm for Cloud Infrastructure	20
3.5 Summary and Discussion	22
4. **Fault Aware Power Efficient Scheduling**	23 – 27
4.1 Power and Fault Aware Reliable Resource Allocation for Cloud Infrastructure	23
4.2 Trust and Deadline-Aware Scheduling Algorithm for Cloud Infrastructure Using Ant Colony Optimization	25
4.3 Summary and Discussion	27
5. **CONCLUSION**	28
6. **REFERENCES**	29 – 30
7. **LIST OF PUBLICATIONS**	31 - 32

1. INTRODUCTION

Cloud computing is a most widespread and popular form of computing, promising high reliability for customers and providers both at same point of time from many field of sciences or industry. Clients from different field are served by datacenters in cloud environment geographically spread over the world. Cloud serves a large number of requests coming from various sources over datacenter with high power consumption. However, to provide such a large computing power required a huge power, leading to high power consumption and cost. Request types in cloud system also affects the services which are public and private requests whose proportion is random in nature. A survey in 2006 over performance of cloud environment in USA shows datacenter consumed 4.5 billion kWh units of power, which is 1.5% of total power consumed in USA and this power requirement is increasing 18% every year [1]. In general cloud computing deals with various issues live poor resource utilization and load balancing and many more. Some of the issues are discoursed as follows: 1) as cloud computing tools are used by industry and they have issues with rapidly growing request and number of servers deployed, increasing the power consumption. 2) Task allocation of request among datacenter with out having knowledge of QoS provided by severs. 3) Current task allocation algorithms only focus on balancing the request and improve unitization of system but not the failure probability of system. 4) High loaded data centers have high failure probability and due to high load this may lead to slow down of datacenter and poor QoS (Quality of service) to the client and client provider. 5) While few of the servers are overloaded and some of them are idle or under loaded. 6) Some request needs to be computed with QoS but due to high load and fault rate they may the QoS promised which is not appropriate to the user and will be a critical issue. 7) As per recent study [4-7], utilization of data centers is a major problem because 60% data centers are idle and most of 20% data centers are utilized and waste of the resources.

This shows poor utilization of resources but this shows the importance of a new approach that has sufficient strategy to minimize wastes of resources and increasing reliability by allocating task over resources which in the case of Cloud is VM with low failure probability to provide high QoS to users. The existing algorithms only take into consideration cloud as non-faulty in nature and fail to provide specific QoS when a fault occurs. So to overcome these issues and improve the performance of the system, we have proposed approaches for resource load balancing and allocation.

1.1. Load Balancing

Load balancing aims to distribute load across multiple resources, such as server, a server cluster, central processing. Load balancing aims to optimize resource use, maximize throughput, minimize response time, and avoid overload of any single resource
Goal of Load Balancing [16] are as follows:

1) To improve the performance substantially.
2) To improve system stability.
3) To have scalability in the system.
4) To improve the system condition under high load or request rate.

Types of Load balancing algorithms [17]
Sender-Initiated: When load balancing algorithm is triggered by the sender.

Receiver Initiated: When load balancing algorithm is triggered by the receiver.
Symmetric: It is the combination of both sender initiated and receiver initiated.

1.2. Resource Allocation

Resource Allocation strategy (RAS) in cloud is all about scheduling of tasks or requests by cloud provider in such a manner to balance the load over all the servers and provide high Quality of Service to clients. It also includes the time required to allocate the resources and the resources available. Main aim is to improve the utilization of resources and complete all the request with in the deadline and with least execution time [2].

An optimal RAS should avoid the following criteria as follows:
a) *Resource contention* situation arises when two applications try to access the same resource at the same time.
b) *Scarcity of resources* arises when there are limited resources.
c) *Resource fragmentation* situation arises when the resources are isolated
d) *Over-provisioning* of resources arises when the application gets surplus resources than the demanded one.
e) *Under-provisioning* of resources occurs when the application is assigned with fewer numbers of resources than the demand.
Resource allocation algorithm can be categorized into three subcategories as from the literature review conducted over existing proposed algorithms.

Categorization is as follows: 1) Static, 2) Dynamic, 3) Learning-based.

Static scheduling algorithms are referred to algorithms which are not affected by system and behavior of cloud some of the algorithms line SJF, FCFS, Round robin etc [24]. On the other hand, dynamic algorithms are those whose objective function depends on the system parameters line deadline, available resources, resource utilization of host and many more example of these algorithms is a deadline-based algorithm, cost-based algorithm, utilization based algorithm [5-9]. Problem with these algorithms is that they do not take into consideration the previous performance of host and system as a whole. Moreover, the past faulty nature of system is not taken into consideration and leads to large request failure. Dynamic algorithms deal with the issue of local minima these algorithms are not able to find global best solution and stuck in local best solution.

1.3. Popular Algorithms

1.3.1. Load Balancing

Various load balancing tactics have been proposed till now which can also be classified into static and dynamic in nature [3]. Yamamoto et al.[4] Proposed a disbursed strategy to balance the load using replication of data. Authors have proposed two replication approaches 1) in the route random replication procedure, replicas saved within the peers along the trail of soliciting for to look. 2) in the course adaptive replication procedure replicas saved most effective within the peers in step with their likelihood of replication. This paper does not use the knowledge about the ability of servers for selection of server.

In [5] Rao et al. have offered a framework for load balancing in distributed environment, named as HiGLOB. Right here, authors have used two principal add-ons 1) histogram supervisor - generates a histogram to preserve a global information regarding the distribution of the load within the system, and 2) load-balancing manager - reallocates the load at any time when the node becomes

overloaded or under loaded. Nevertheless, there's overhead associated at the same time setting up and preserving the histograms.

Zeng et al. [6] have proposed a load re-balancing algorithm to work out the crisis of load balancing in distributed environment. They have additionally ensured the reliability of the process where one chunk of a file and two duplicate copies are allocated in three exceptional different servers at a time. In this algorithm author uses the master server periodically for checking of chunk servers and to differentiate which chunk server is over- loaded and which is not. Nonetheless, this master server turns into a single-point failure for the load balancer.

Fan et al. [7] have proposed an adaptive load optimization algorithm (AFLBA) for the Hadoop distributed file procedure which uses two modes: 1) disk utilization expense system and 2) carrier blocking off rate system. The proposed algorithm uses the storage average utilization of each data node and probability of blocking consumer request of each knowledge node. Since this algorithm isn't disbursed so it creates a performance bottleneck node within the HDFS.

Hasio et al. [1] and Chung et al [8] have proposed an improved load balancing algorithm for distributed file system to overcome the issue of bottleneck and improve the performance of system. They have proposed to use CHORD protocol for creation the node server.

In [11-15] authors have presented various improved ant colony based load balancing algorithms for cloud to over the drawbacks for previously proposed algorithms.

1.3.2. Resource Allocation

Many researchers have proposed various ways to optimize scheduling in cloud environment and improve the performance. Some of them are listed below. [18] Proposed an Min-min algorithm, in this algorithm a task with least execution time and task allocation is allocated accordingly. Proposed load balance algorithm have all characteristics of Min-min algorithm and improves the performance as compared to FCFS and round robin algorithm..

Hu, Jinhua et al. [19] proposed a scheduling algorithm based on collective and collaborative computing of trust model. Here the trust value is the main factor to schedule the task, where trust value is manually taken from both client and service provider, which makes is free from all computational process. Author has proposed a way to improve the system reliability and evaluate the trust worthiness of VM and evaluate the trust value.

Hu, Jinhua et al. [20] proposed and algorithm for load balancing using live VM migration. VM migration is on the basis of load over a host, if the load over a host is more then a threshold during the execution of request. If the host gets over loaded a VM is selected to be migrated to other under loaded host.

While live migration of VM resources are reserved on both the host i.e source and destination in such a manner that CAP (maximum availability of CPU) is taken care of and adjusting memory resource dynamically.

Barroso et al. [21] – proposed an improved ACO (ant colony optimization) based scheduling algorithm to balance the request in cloud computing environment. This paper proposed LBACO for request load balancing in distributed environment.

Xiaobo et al. [22] proposed n real line VM allocation model, based on energy efficiency model to schedule with minimum cost using Rt-VM policy for VM scheduling.

Suraj, S. Rin et al. [23] have proposed a learning based task allocation using genetic algorithm to find schedule with least execution time and high resource utilization in cloud environment.

1.4.Parameters Used

Fault rate: defined as the total count of request failed over a period of time T

Failure Probability: as the probability of request to fail on a specific host or system.

Reliability of a system: This feature of a system can derived from the failure probability of system which can be defined as:

$$Reliability\% = 100 - Failure_Probability\%$$

Power Efficiency: The ratio of the output power over the input power i.e. the percentage power consumed over a period of time.

Utilization: this is the capability of host to be used out of total available resources.

Average Resource Utilization: This is an average of utilization of resources over the whole system i.e. all available hosts.

Average start time: Average waiting time of request before been scheduled or allocated.

Average Finish time: Average of finishing time of all the request executed by system.

Scheduling Delay: Total time to find a suitable resource for a set of tasks.

MakeSpan: Total execution time of system including scheduling delay for set of requests /Tasks

2. FAULT AND DEADLINE AWARE LOAD BALANCING

With the rapid growth in technology, there is a huge proliferation of data requests in cyberspace. Distributed system/servers plays a crucial role in the management of request in cloud which are distributed among the various geographical zones. Many of time the system gets over loaded due to few of servers with high number of request and some of servers being idle. This leads to degradation of performance of over loaded servers and failure of requests. On these over loaded servers average response time of server increases. So there is a requirement to design a load balancing algorithm to optimize resource utilization, response time and avoid overload on any single resource

In this section, we have proposed a set of approaches that balances the load of servers and effectively utilizes the server capabilities and resources. The main contribution of this work is to improve the average resource utilization of system and removing hot spots and cold spots in the system i.e. the unbalancing of requests over the system should be removed.

2.1 Approaches 1: Fault and Load Aware Load Balancing in Cloud Storage

Here, we have proposed a Fault and Load based Load balancing algorithm (FLBLBA) that can balance a load of servers dynamically by considering its parallel processing capability, processing time and its request queuing capacity. The proposed approach takes four main parameters of a server 1) Server request queue size - buffer space to store the client requests to be handled by the server. 2) Server service rate (λ) - the number of CPUs available for processing the client request in a server. 3) Processing time (S_T) – time takes to process a request which differs from server to server.4) Fault rate. Modern servers are equipped with many features like multiple CPUs, large storage, high I/O capability etc. We have chosen the multiple CPUs feature as a main parameter for load balancing of our proposed approach.

2.1.1. Proposed Model

Proposed algorithms have been designed to balance the client requests over the servers and distribute the load over the system uniformly. Here, load balancer as shown in Fig. 2.1.(a) regularly exams for the request queue dimension of server and tries to restrict the problem of overloading of any server with the aid of migrating the extra request to other idle or least loaded and least faulty neighboring server in cloud.Proposed load balancing algorithm is divided into two stages. In first stage list of idle servers is created, and in second stage the server with highest fitness value and which can fulfill the request with least failure probability.

Stage I

Algorithms checks and calculate the fitness value for the neighbor server to store them in a list shown in Fig. 2.1(b). Load balancer utilizes this list to select the server that has highest fitness value. Load balancer calculates the waiting time over each server from above list which can be given as:

$$W_k = \frac{Q_L_{current_k}}{\lambda_k} \times S_T_k \quad (1.1)$$

$$F_j = \alpha_1 .\Delta Mj + \alpha_2.\lambda_j + \alpha_3.(\frac{1}{FR_j}) + \alpha_4 .(\frac{1}{W_j}) \quad (1.2)$$

$$\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 = 1 \quad (1.3)$$

Equation 1 shows the W_k waiting time of i^{th} request at server 'k'.

Stage II

In second stage load balancer then finds the server with least waiting time, least fault rate and highest service rate i.e. highest fitness value from the list.

A proposed algorithm also tries to improve the server response time by selecting the server having least CPU utilization. In this way, proposed algorithm utilizes the idle or underutilized server to increase the overall performance of the system and reduce requests failure over the system by reducing the probability of request failure.

Algorithm : Load balancing

1. **FLBLBA** (Server s, $Q_L_{current}$,λ_k,S_T_k, FR_i)
Input: Server s, Queue length $Q_L_{current}$, service rate λ_k, service time S_T_k, fault rate FR_i
2. s← server
3. $Q_L_{current}$ ← current queue size
4. λ_k ← Service rate of server k
5. S_T_k← Service time of server k
6. FR_i ← fault rate of server k
7. Compute W_k
8. **if** ($\delta\rho$> $Q_L_{Threshold}$) **then**
9. check server queue status.
10. Add request to queue;
11. Process_request();
12. **else**
13. server is overloaded
14. S ← Find_server(server_neighbour_list L)
15. find under loaded server.
16. S ← migrate request
17. Goto step 7

Output: Load balances the request.

Algorithm : To find suitable server

1. **Find_server**(server_neighbour_list L)
Input: server_neighbour_list L
2. **for** k=1 to L.size()
3. S1←L.get()
4. F_k= **α1** ΔMj + **α2**λ + **α3** (1/FR_j)+ **α4** (1/W_j)
5. temp_list t← F_k
6. **end for**
7. L2 = Sort(t)
8. S2 ←min (L2)
9. return S2

Output: The server with minimum fitness value.

Fig. 2.1. Proposed algorithms (a) Load balancing algorithm. (b) Find neighbor server algorithm

λ_i - Service rate that is number of request processed simultaneously server.
S_T -Service time is the time taken by server to process the request
$Q_L_{current_k}$ -Current queue length of server
$Q_L_{threshold}$ - Threshold limit of server request queue.
W_j - Waiting time for a request at server j.
Fr_j -Fault rate that is the number of request failed due to system failure over time t.

F_j - Fitness value of neighbors of server j. (j $\exists$ {1,2,3,….. N}

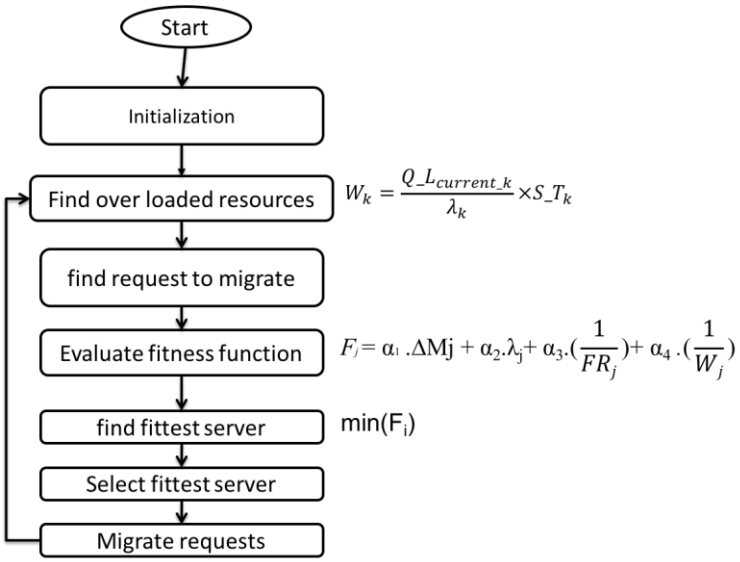


Fig. 2.1. Proposed algorithms (c)

2.1.2. Experiment and Result

Performance analysis of proposed FLBLBA algorithm is finished using cloudsim simulator where we now have thousands of requests to be completed by 12 storage servers. All of the servers work concurrently with constant quantity of CPU cores to process the client request rapidly. Each server has a request queue to buffer the incoming client requests, storage ability to store the data and fulfill the client requests.

Table 1. Experimental parameters used for simulation environment

No. of client requests	No. of Servers	No. of CPU cores available per server	Storage capacity of servers (GB)	Server queue length
800	12	7	500	20
1000	12	7	500	20
1200	12	7	500	25
1800	12	7	500	25
2400	12	7	500	25

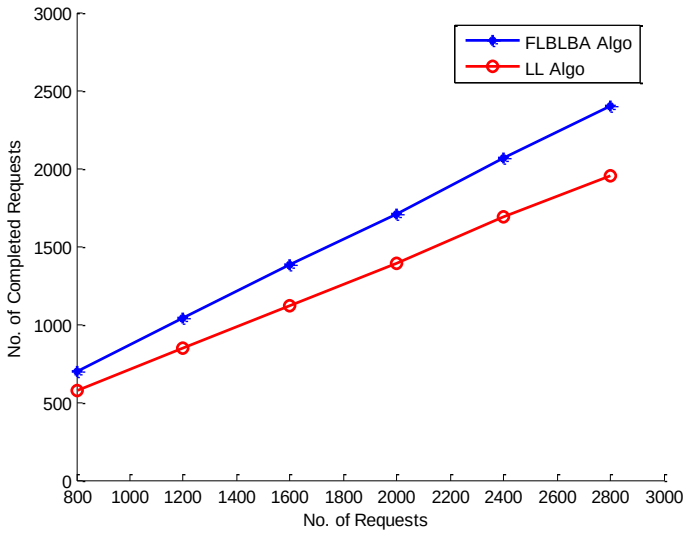


Fig 2.2 Number of request completed

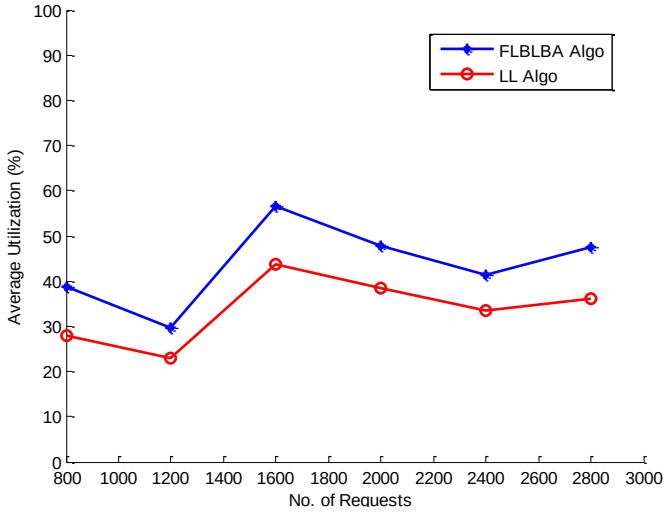


Fig. 2.3 Average utilization of system

Figure 2.2 shows improvement in number of completed request using proposed algorithm. Figure 2.3 shows the improvement in average utilization using least loaded and proposed FLBLBA reflecting improvement in load over system.

2.2 Approaches 2: Deadline Aware Load Balancing of Distributed Servers in Distributed

Here, we have proposed a Deadline based Load balancing algorithm (DDLBA) that can balance the load of servers dynamically by considering its parallel processing capability, processing time and its request queuing capacity.

2.2.1. Proposed Model

Proposed algorithms have been designed to balance the client requests over the servers and distribute the load over the system uniformly. Here, load balancer as shown in Fig. 2.4(a) is responsible for consistently assessment of the request queue dimension of servers and tries to prevent the main issue of overloading of any server with the aid of

migrating the request to other idle or least loaded neighboring servers which can also complete the request in deadline without failure in the system. Proposed load balancing algorithm is divided into two stages. In first stage list of idle servers is created, and in second stage the server with highest fitness value and which can fulfil the deadline in selected.

Stage I

Algorithms checks and calculate the fitness value for the neighbour server to store them in a list shown in Fig. 2.4(b). Load balancer utilizes this list to select the server that has highest fitness value.

Stage II

In second stage load balancer calculates the waiting time over each server from above list which can be given as:

$$W_k = \frac{Q_L_{current_k}}{\lambda_k} \times S_T_k \quad (2.1)$$

$$W_k > DL_i \quad (2.2)$$

Equation 2.1 shows the waiting time of k^{th} request. DLi deadline of a request. Load balancer then finds the server with waiting time less than request deadline and highest fitness value from the list. Proposed algorithms also try to reduce the server response time by selecting the server with least CPU utilization. In this way, we utilize the server to increase the overall performance of the system and reduce request deadline failures over the system.

Algorithm : Load balancing

1. **DDLBA** (Server s, $Q_L_{current}$, λ_k, , S_T_k, DLi)
Input: Server s, Queue length $Q_L_{current}$, service rate λ_k, service time S_T_k, deadline DL_i
2. s← server
3. $Q_L_{current}$ ← current queue size
4. λ_k ← Service rate of server k
5. S_T_k← Service time of server k
6. DL_i← Deadline time of request I
7. Compute W_k
8. **if** ($\delta\rho > Q_L_{Threshold}$) **then**
9. check server queue status.
10. Add request to queue;
11. Process_request();
12. **else**
13. server is overloaded
14. S ← Find_server(server_neighbour_list L)
15. find under loaded server.
16. S ← migrate request
17. Goto step 7

Output: Load balances the request.

Algorithm : To find suitable server

1. **Find_server**(server_neighbour_list L)
Input: server_neighbour_list L
2. **for** k=1 to L.size()
3. S1←L.get()
4. F_k= α1 ΔMj + α2λ
5. temp_list t← F_k
6. L2 = Sort(t)
7. **end for**
8. **for** j=1 to L2.size()
9. if(L2. W_k< DL_i) then
10. S2 ←L2
11. **end for**
12. select server which fits the deadline and have maximum fitness value.
13. return S2

Output: The server with minimum fitness value.

Fig. 2.4. (a) Load balancing algorithm. **(b) Find neighbor server algorithm**

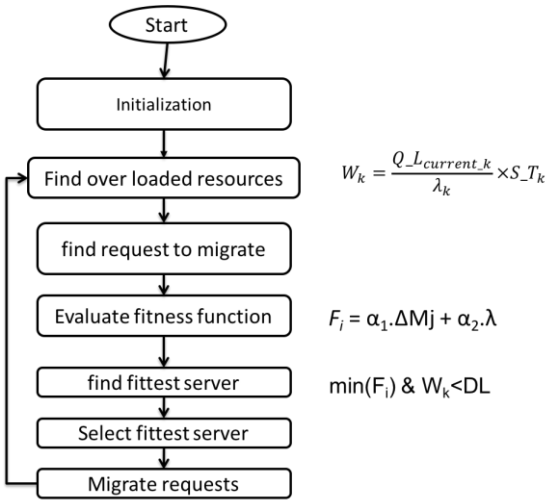
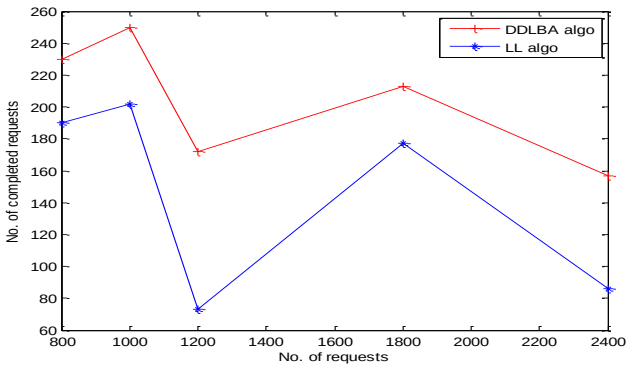


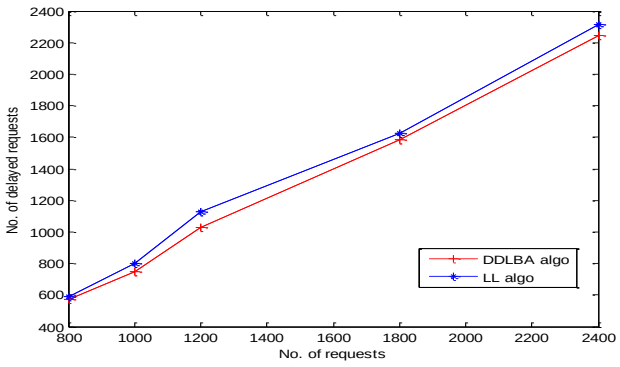
Fig. 2.4. (c) Proposed algorithm

2.2.2. Experiment and Result

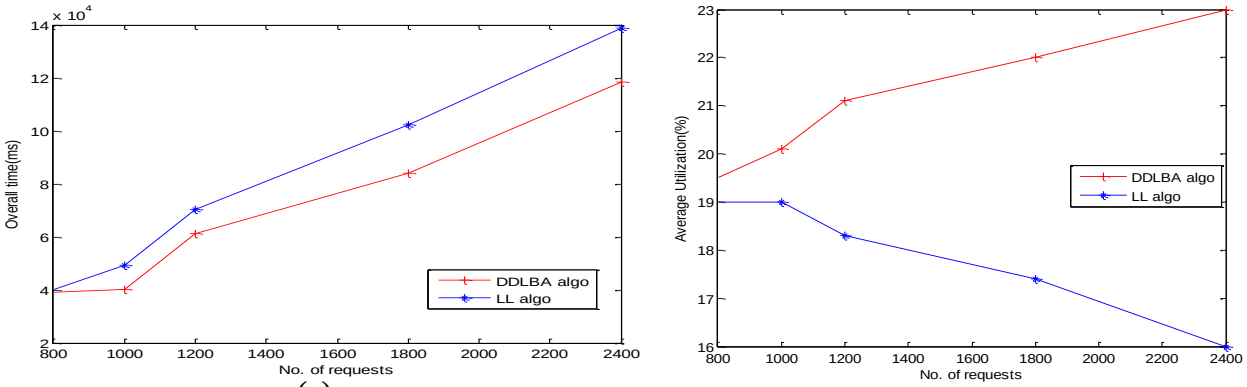
Efficiency evaluation of proposed DDLBA algorithm is completed utilising simulations the place we've created hundreds of thousands of virtual machine requests to be served via 12 storage servers. All servers are deployed in distributed environment with unique count of CPU cores complete the requests. Each server has a fixed request queue size to buffer the incoming client requests and storage capacity. For the given main issue assertion in part 3 where the load is unbalanced, it is assumed that half of storage servers get client requests and others stay idle. Our aim is to equally distribute the received requests among the many servers to restrict the scenario of overloading. Within the simulation situation numbers of storage servers are kept fixed with various quantity of requests handling. We've got additionally compared the bought results with the least load balancing algorithm. Following table depicts the configuration parameter for our simulation atmosphere.



(a)



(b)



(c) (d)

Fig. 2.5. Processed client requests (a) no. of sent requests vs. no. of completed requests. (b) no. of sent requests vs. no. of postponed requests. (c) Overall response time. (d) Average utilization of system

2.3 Approaches 3: Load Balancing Algorithm for Hybrid Cloud IaaS

The Trust Model we are proposing here will work for Private, Public and Hybrid Cloud. It will take into consideration both the Direct and Indirect Trust or recommended trust. For the Calculation of Trust Value we will take Memory (RAM), MIPS (Million Instruction per Cycle), Frequency (Frequency of data center) and Fault Rate. We will initially calculate only Direct Trust and as the time will evolve we will also consider the indirect trust. We will initiate the parameter MIPS and Fault with zero and Memory and MIPS will be according to the datacenter. With these we will calculate the Trust Value and initial load balancing will be done according to that. As the Time will evolve, we will calculate the Indirect or Recommended Trust also using Fault Rate which we initially considered zero and Response Time. Now, since the request from both public and private cloud will start arriving, we assume that some of them will not be accomplished due to technical faults and hence the parameter Fault Rate will have some value other than zero.

Meanwhile calculate the Indirect Trust, which is based on:

Fault Rate (number of task failed/total tasks)
Response Time (Time taken to accomplish a standard request.)
RAM: flash memory of the server.
MIPS: Millions of instructions per second of server.
Initial_Trust : Trust value of the server based on the executional power i.e. RAM and MIPS.
Fault_Rate : Number of faults in an server over a period of time 't'.
Updated_Trust : Updated Trust value based on initial trust and fault occurred over a period of time.
α_1, α_2 & α_3: Constants

$$\text{Initial_Trust} = \alpha_1 * \text{RAM} + \alpha_2 * \text{MIPS}; \quad (3.1)$$

Where

$$\alpha_1 + \alpha_2 + \alpha_3 = 1 \quad (3.2)$$

$$Updated\ Trust = InitialTrust + \alpha_3 * (\frac{1}{FaultRate}) \quad (3.3)$$

Algorithm : Load Balancing Algorithm

```
1. Load Balancing Algorithm(Req_list r)
   Input: Requests list r
2. Initialize servers
3. Calculate individual trust values of each data center
4. CheckDirect _TrustValue ( )
5. CheckIndirect _Value ( )
6. dividing them into public or private according the size of
   request.
7.  if (Data Center Value > Threshold) then
8.        Send the private cloud requests.
9.  else if (Data Center Value < Threshold)
10.       Send the public cloud requests.
11.  else
12. Keep Searching
Output: All request been scheduled.
```

Fig. 2.6 Proposed algorithm

2.3.1. Experiment and Results

Proposed trust model is simulated using Cloudsim API [24]. Cloudsim basically support cost estimation, and Random Load balancing of resource and has no support to study behaviour of datacenter and trust model. So to study the datacenter behaviour trust parameter is introduces as datacenter property, which depends on parameter defined in Cloudsim. Based on this attribute we have computed the result to show the real problem. For this we have considered two classes of budget range, with 3 datacenters and 300 user requests.

Table 2

Servers Parameters

Server Name	Fault rate	Hardisk (MB)	RAM (MB)	MIPS	Cores
Server1	0.143	1000000	20048	10000	4
Server2	0.125	1000000	20048	10000	6
Server3	0.5	1000000	20048	10000	4

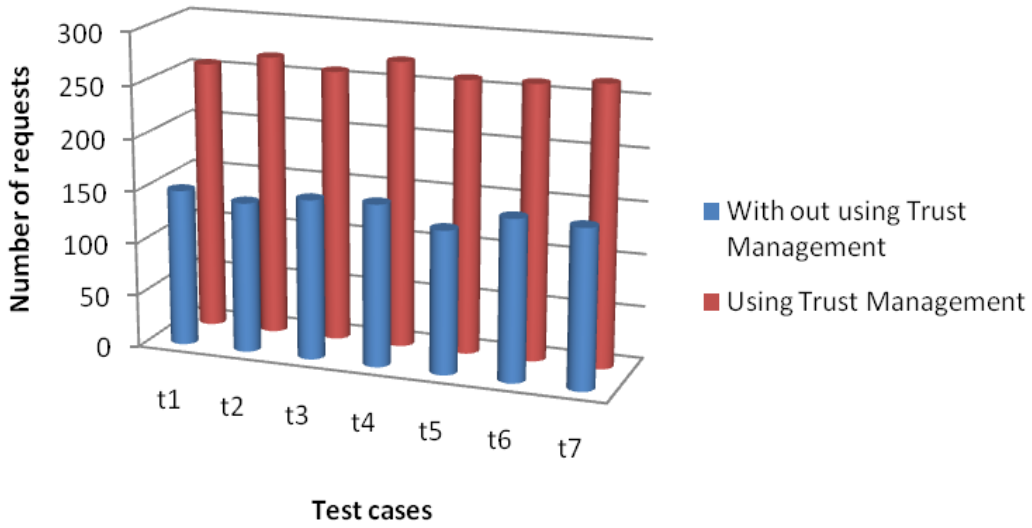


Fig. 2.7. Total number of request completed in faulty environment

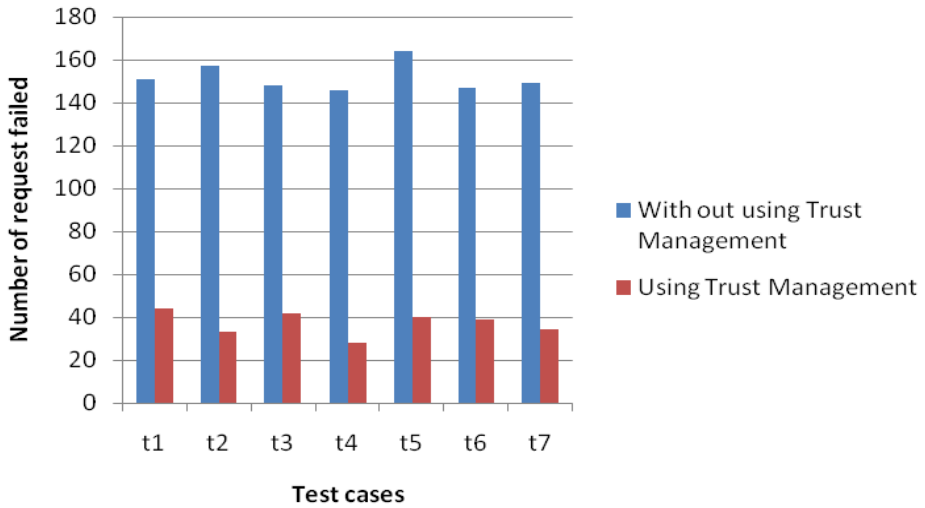


Fig. 2.8. Total number of request Failed in faulty

Figure 2.7 and 2.8 represents the improvement in request failure and completed count using proposed algorithm as compared to existing algorithm with out trust management.

2.4. Summary and Discussion

The main achievement of this study and work was to study the rich literature and solve the issue of load balancing in fault aware cloud environment. The results obtained with our approach were very competitive with most of the well known algorithms in the literature and justified over the large collection of requests. Proposed load balancing algorithm proves to provide better fault tolerance as compared to existing algorithm with least request failure, reduced average utilization, average delay and high request completion count.

3. LEARNING BASED FAULT WARE RESOURCE ALLOCATION

Distributed and cloud computing is presently slanting and more prevalent in nowadays for the computation by numerous organizations like Google, Amazon, Microsoft and so forth. As the cloud size increments with expansion in number of server's power consumption over a server farm increments. Also with increase in request load over a datacenter increases the request failure probability. To over come these issues request should be scheduled in more efficient manner, improving resource utilization, request failure count and reliability of system. Recent studies show that the failure probability of a server, increasing in a linear way due to increase in number of independent resource (processors) resulting in request failure at datacenters. So to overcome these issues the proposal are made and to improve the performance.

In this section we have proposing set of learning based algorithm for task allocation to minimize the request failure and improve QoS (Quality of Service) over a data center. Proposed approached aims to provide a global best schedule with least scheduling time complexity. Proposed algorithms has proven to have better performance in term of load and request failure rate as compared to previously proposed task allocation algorithm for cloud IaaS.

3.1. Approaches 1: Fault and QoS based Genetic Algorithm for Task Allocation in Cloud Infrastructure

To overcome these issues a fault aware learning based resource allocation algorithm is been proposed using Genetic algorithm. Genetic algorithm helps us to find a solution, which cannot be, achieved my any static or dynamic algorithm. More over fault tolerant genetic algorithm help to find a fittest solution in term of least makespan (Time taken to complete a request) and least request failure probability. Proposed algorithm uses Poisson probability distribution for random request failure at virtual machine i.e. at host and datacenter level. On the other hand, request failure over a datacenter may occur randomly due to storage, network failure or VM crashes. Based on fault over a datacenter and computing capability of a system, we have proposed a task allocation policy to minimize the total makespan over the system and reduce request failure probability. According to algorithm collect the information of data center resources and capability, and the count of failure occurred over a period of time on a datacenter.

Proposed algorithm is divided into four phases which are as follows:

I. Initialization
II. Evaluation and selection

III. Crossover

IV. Mutation

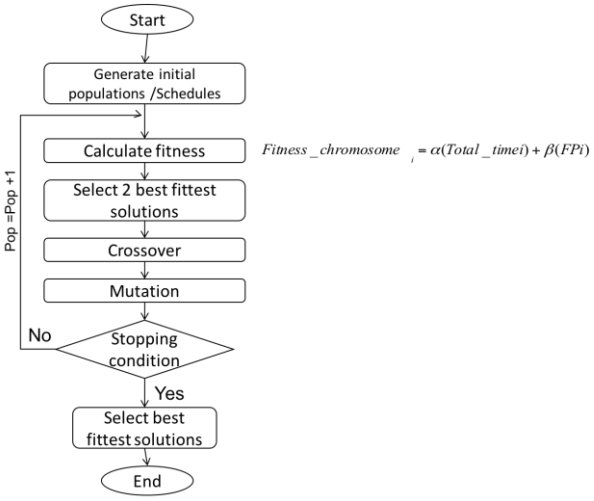


Fig.3.1 Proposed genetic algorithm

3.1.1. Experiment and Results

For simulation CloudSim 3.0 API is used. CloudSim 3.0 provides linear power model simulation to find the power consumption in cloud.

Proposed fault aware genetic task allocation algorithm is implemented in CloudSim replacing existing task scheduling to find the global best schedule. Proposed algorithm is being tested over various test cases with 10 servers D0-D9 and Poisson distribution model for random request and fault model in distributed environment.

Testing of proposed algorithm is done with basic Genetic algorithm proposed by Suraj, S. Rin [24]. Testing is done for 1000, 1500, 2000, 2500, 3000, 3500 requests with population size been 100, 200, 300, 400. Iteration for simulation of each simulation is 100. Results are shown in 3.2-3.5

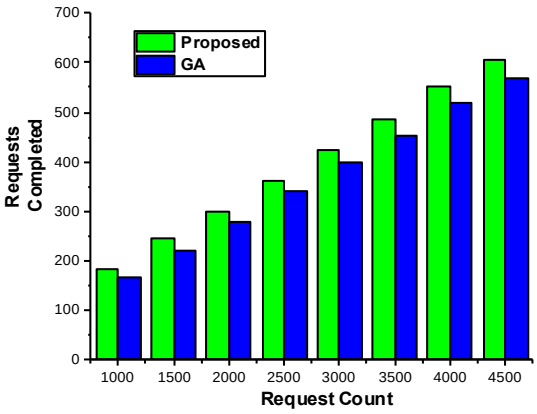


Fig.3.2 Comparison of improvement in request completed

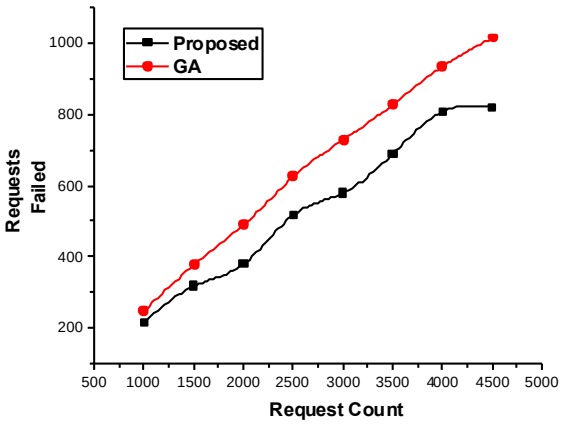


Fig.3.3 Comparison of improvement in request failed

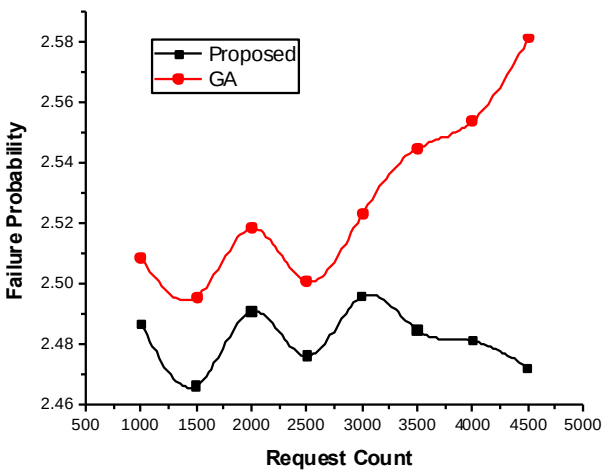


Fig.3.4 Comparison of failure probability with variable request

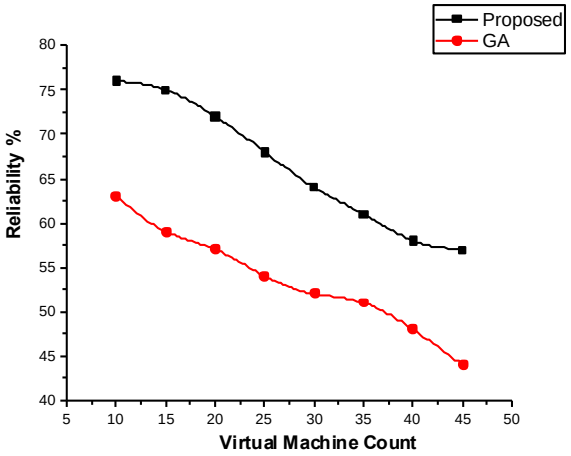


Fig.3.5 Comparison of reliability with variable resources

3.2. Approaches 2: Task Allocation Using Big Bang-Big Crunch in Cloud Infrastructure

We have proposed a task allocation algorithm based on Big Bang-Big Crunch (BBC) algorithm. This algorithm is motivated from the physics behind creation of universe theory in astrology. BBC algorithms refers to the evolution of universe and end of universe which says universe is a finite space which once expanded with force binding it and will end into a single point referred as a black hole. Algorithm also suggests that any element of universe cannot be suggested as center of universe. Similar to this we have proposed a task allocation algorithm to find a single best solution from a large set of solutions. Where generation of universe is referred as Big Bang phase and dissipation of universe in black hole near the center is said to big crunch phase.

Proposed algorithm uses Poisson probability distribution for random request at virtual machine i.e. at host and datacenter level. Based on computing capability of a system, we have proposed a task allocation policy to minimize the total makespan over the system and reduce time complexity of solution. According to algorithm collect the information of data center resources and its capability. Proposed algorithm is similar to Genetic algorithm (GA) but the problem size reduces after each phase and will give you a single point solution i.e. the global solution. But in existing GA the population size remain same and there is no guarantee that the global best is achieved.

3.2.1. Proposed algorithm

Proposed algorithm is divided into four phases which are as follows:

I. Big Bang /Initialization phase

II. Evaluation phase

III. Crossover / Center of mass

IV. Big Crunch phase

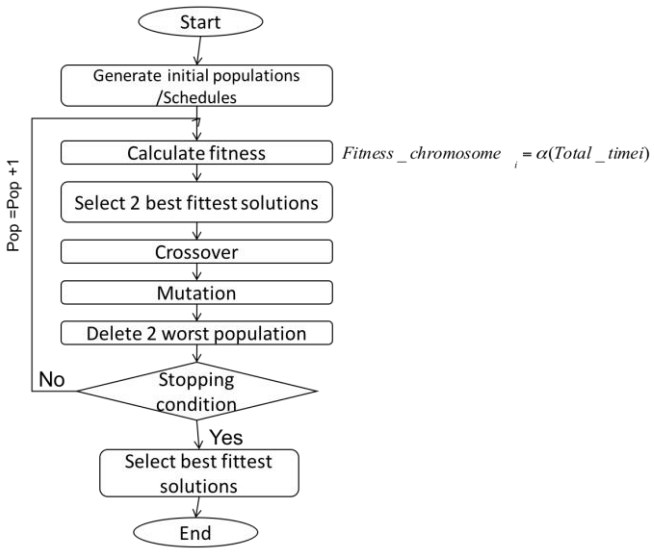


Fig.3.6 Proposed big bang big crunch algorithm

3.2.1. Experiment and results

Testing of proposed algorithm is done with basic Genetic algorithm with test suite same as in section 3.1.1.

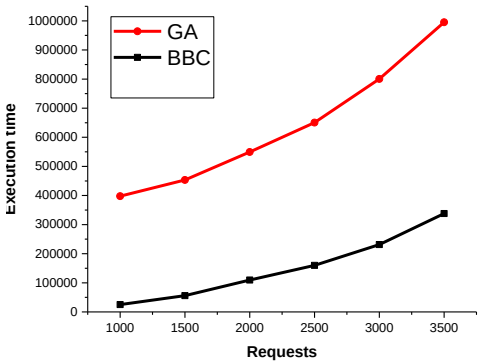


Fig.3.7 Comparison of improvement in execution time

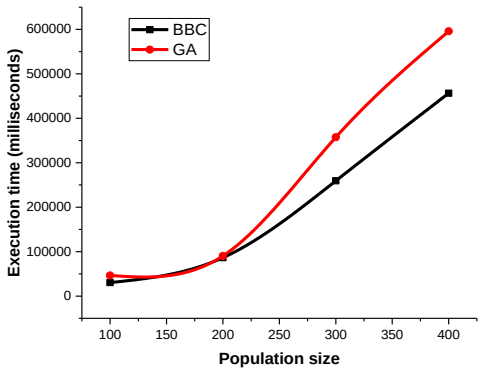


Fig.3.8 Comparison of improvement in execution time with changes in population size.

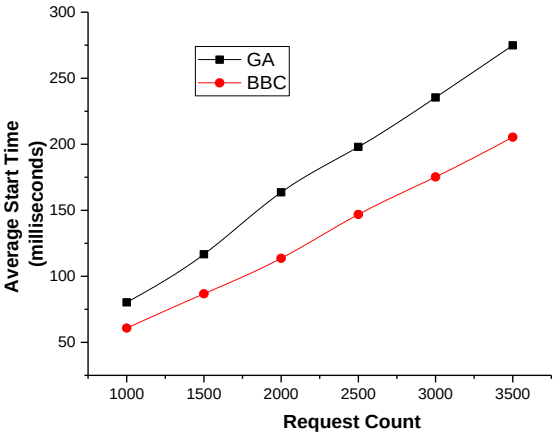


Fig. 3.9 Comparison of Average Start time of system with increase in request count

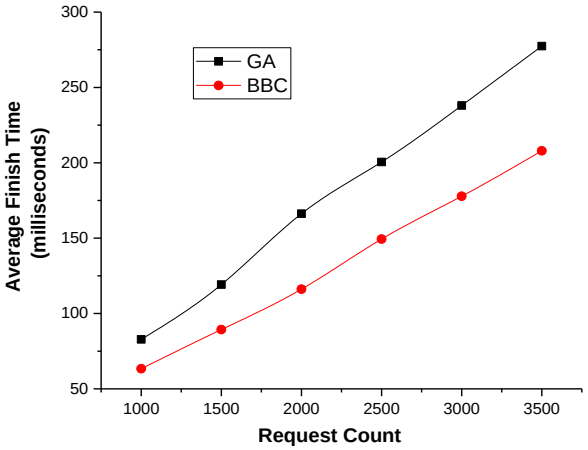


Fig.3.10 Comparison of Average Finish time of system with increase in request count

3.3. Approaches 3: Fault Tolerant Big Bang-Big Crunch for Task Allocation in Cloud Infrastructure

3.3.1. Proposed algorithm

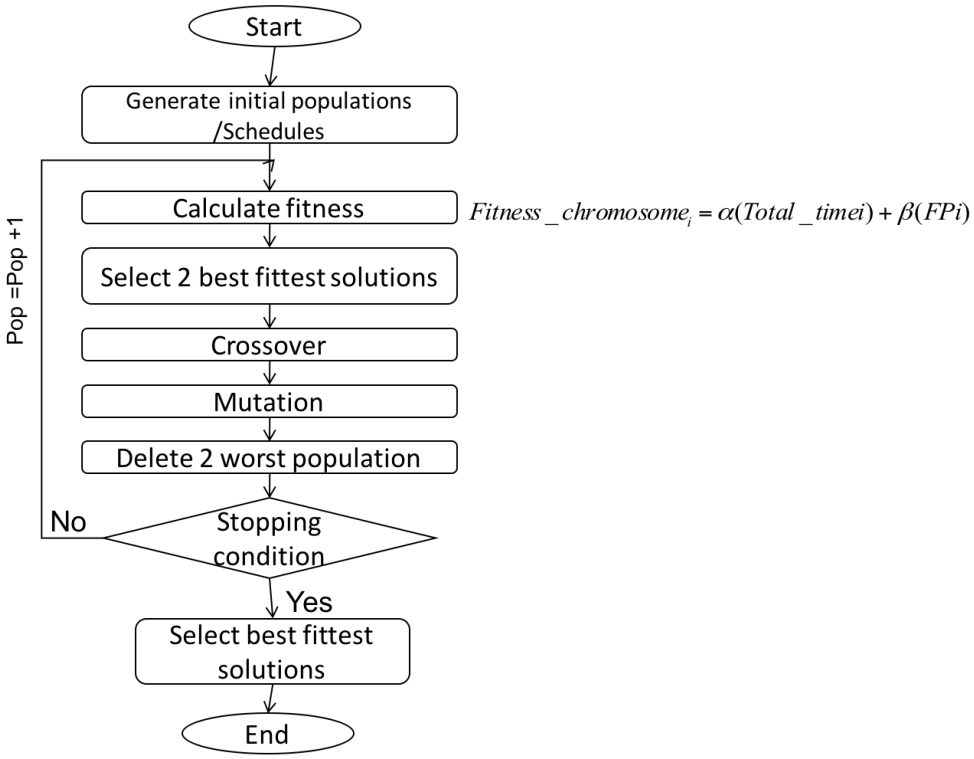


Fig.3.11 Proposed fault aware big bang big crunch algorithm

3.3.2. Experiment and Results

Testing of proposed algorithm is done with basic Genetic algorithm with test suite same as in section 3.1.1.

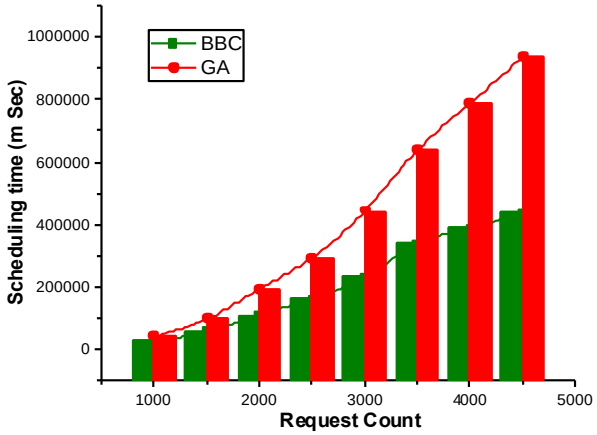


Fig.3.12 Comparison of improvement in scheduling time

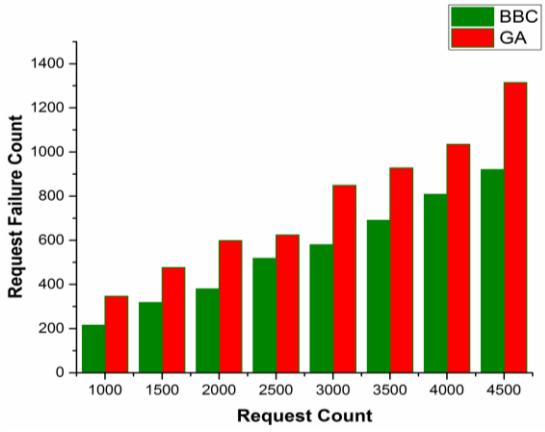


Fig.3.13 Comparison of improvement in request failed

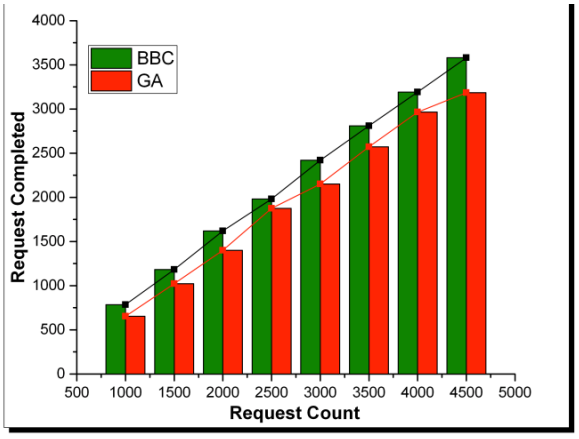


Fig.3.14 Comparison of improvement in request completed

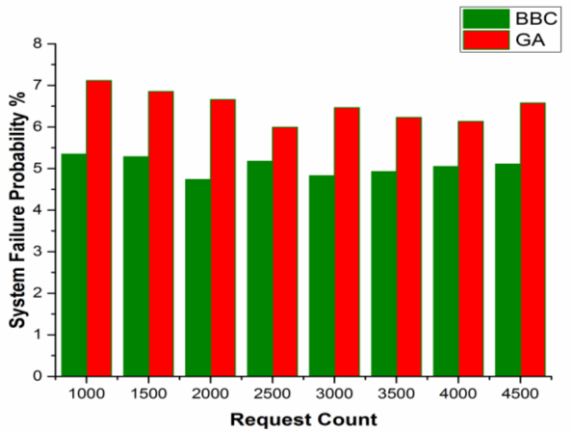


Fig.3.15 Comparison of failure probability with variable request count

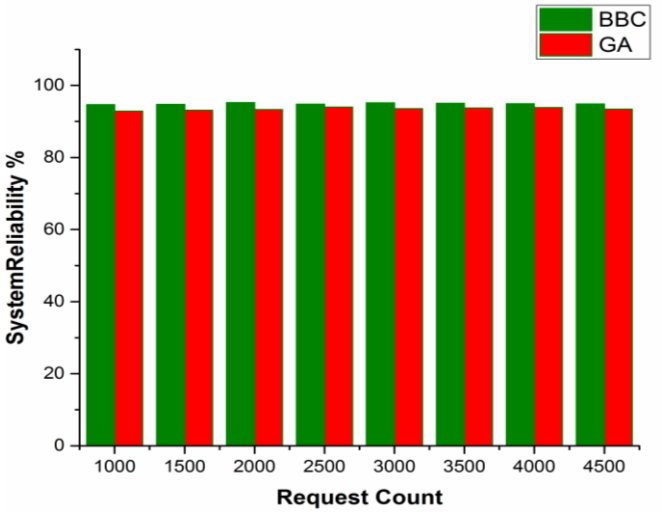


Fig.3.16 Comparison of reliability with variable request count

3.4. Approaches 4: Load and Fault Aware Honey Bee Scheduling Algorithm for Cloud Infrastructure

There exist many load balancing algorithms proposed for grid and distributed computing environments [24]. But they do not take into consideration cloud as non faulty and QoS of datacenter. So to overcome this issue and make system more reliable, fault and load aware honey bee scheduling algorithm ins proposed.

Proposed algorithm is inspired for natural behaviour of honey bee to find the best best solution for designing optimal scheduling algorithm. The algorithm requires a number of parameters to be set, specifically: quantity of scout bees (n), quantity of nice websites out of m selected sites (e), number of websites selected out of n visited sites (m), number of bees recruited for high-quality e web sites, number of bees recruited for the opposite (m-e) selected sites, initial size of patches which includes site and its neighborhood and stopping criterion

Fitness (FT), allocated MIPS (MP), Fault rate (FR), Initialization time (IT), Network load (N_L).

(3.1) $$FT = \alpha_1 \frac{1}{N\,L} + \alpha_2 \frac{1}{FR}$$

$\alpha_1<1$, $\alpha_2<1$ & $\alpha_3<1$ (3.2)

$\alpha_1 +\alpha_2 +\alpha_3=1$ (3.3)

Fault rate (FR) is:

$$FR(t)= f(MP , N_L) \quad (3.4)$$

Where FR(t) is number of faults over the time t, which is function of system and network load over the time t. α_1, α_2 and α_3 are constant which represents the ratio of parameters contribution to fitness value.

Algorithm :Honey bee allocation

1. Honey_bee_allocation(Req_list r)

Input: Requests list r

2. Initialize scout bees equal to number of datacenters.
3. Recruit scout bees for selected sites (more bees for best e sites)
4. Evaluate fitness value for datacenter.
5. Assign scot bees to search randomly and evaluate their fitness for a request.
6. Stop when all bees have arrived, else wait.
7. Select the fittest bee
8. Assign remaining bees to search randomly and evaluate their fitness for each request.
9. End While no request in queue.

Output: All request been scheduled.

Fig.3.28 Proposed Algorithm

3.4.1 Experiment and Result

Proposed fault aware honey bee algorithm is simulated using CloudSim 2.0 simulator [24]. CloudSim originally support Round robin, cost based algorithm, and FIFO algorithm for scheduling the resource sequentially. Originally CloudSim 2.0 API does not support faults in cloud environment. So firstly occurrence of fault is added as a parameter for datacenter which responds to failure probability of the datacenter.

This CloudSim API is used to set up a cloud infrastructure environment for simulation. So that environment includes all cloud IaaS request functions and environmental parameters, host and datacenter parameters. Proposed algorithm is implemented in cloudsim changing existing algorithm to study and improve the performance. Comparative study is made between basic load aware honey bee (BLHB) and proposed algorithm.

Table 4

Server Name	Fault rate FR(t)
Server1	0.143
Server2	0.125
Server3	0.5

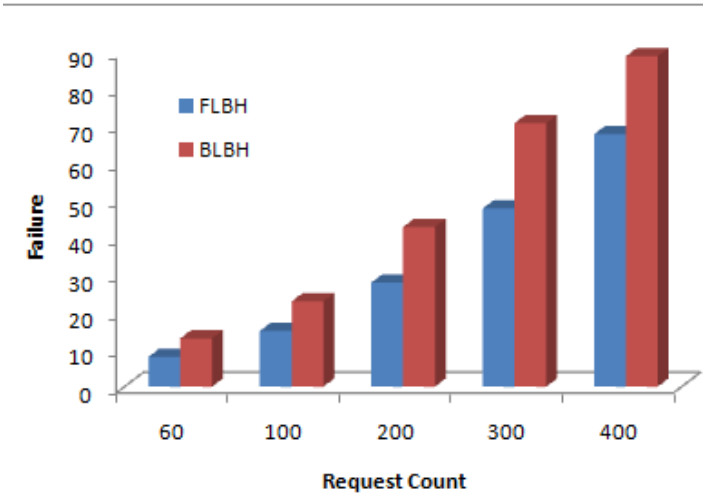


Fig. 3.29 Comparison of request failure count.

3.5. Summary and Discussion

The main achievement of this study and work was to study the rich literature and solve the issue of resource allocation in fault aware cloud environment. The results obtained with our approach were very competitive with most of the well known algorithms in the literature and justified over the large collection of requests. Proposed resource allocation algorithm proves to provide better fault tolerance as compared to existing algorithm with least request failure, reduced average utilization, Scheduling delay, high request completion count, Failure probability and improved reliability of system.

4. FAULT AWARE POWER EFFICIENT SCHEDULING

A Cloud computing is now a trending way of computing tasks and more general these days. Cloud computing is adopted by many firms like Google, Amazon, Microsoft and many more for reliable and efficient computing. But as the cloud size increases with expand in number of datacenters and vigorously increasing consumption of power over a data center. Also with increasing request load over a server the computing load on server's increases, leading high power consumption. So there is a need to balance the request load in such a manner to effectively improve the resource utilization, load with reducing request failure and power consumption.

Cloud computing has made it complicated with variable length request whose proportion may increase or decrease effecting the cloud. Recent surveys show that the power consumption of a datacenter increases linearly with increase in utilization due to request load over a datacenter. This results in high request failure and decreasing power efficiency of system. Resource allocation done without having knowledge of load and power efficiency of a datacenter will increase the power consumption of system and high request failure count. So to over come these issues a fault and power aware resource allocation and scheduling algorithms are proposed to improve the power efficiency, failure count and average load over a datacenter. Proposed algorithms shows improved performance in term of average load and power efficiency as compared to existing algorithms for cloud infrastructure.

4.1. Approaches 1:Power and fault aware reliable resource allocation for cloud infrastructure

The problem with existing algorithm is that they used for simple task scheduling to improve resource utilization or power efficiency in cloud and manage quality of service of a datacenter. Existing algorithms also assumes cloud as non faulty in nature so do not takes fault occurring in the system for scheduling and only taken load over da datacenter, which is in sufficient to provide better QoS to the user. So to conquer these issues a power and fault aware resource allocation algorithm is proposed. Proposed algorithm utilizes linear power model or evaluation of power efficiency of datacenter. Failures over a datacenter occurs randomly that may be due to network or storage failure. Proposed VM allocation algorithm aims to minimize the power consumption of the system and reduce request failure count.

Parameters to evaluate fitness value:
PD_i: i^{th} Data center.
PE_i : power efficiency of i^{th} host in a data center .
U_i : Current Utilization of i^{th} host in a data center.
FR_i : Fault rate that is the number of request failed due to System failure over time t.
FP_i : Failure probability over a Host i.
F_i : Fitness value of i^{th} host.

//Fitness value $$Fi = PEi + FPi(t) \quad (11)$$

4.1.1. Proposed algorithm

Algorithm :PFARA

1. PFARA (PD, Q_{length})

Input: Datacenter List PD and Queue length Q_{length}

2. PD← Host List
3. i ←No. of Data Centers
4. Q_{length}← current queue size
5. PEi ← Power Efficiency of Host Pd_i
6. Fpi ← Failure Probability of Host Pd_i
7. If(Q_{length} !=0) then
8. Allocate_Resources (Req)

Output: All request scheduled.

Fig. 4.1 Proposed Algorithm initialization

Algorithm : Find fittest host

1. Allocate_Resource(Req r)

Input: Requests list r

2. Host_list= gethostlist();
3. Sorted_host = Sort_Fitness (Host_list)
4. **for** (Host h: Sorted_host)
5. Fi=PEi + FPi (t);
6. If(h.isSuitable() && h.fitness_Least())
7. selected_host= h;
8. **end for**
9. **if**(selected_host != NULL) **then**
10. allocate(request, selected_host)
11. **else**
12. printf("cannot find suitable server")

Output: The server with minimum fitness value.

Fig. 4.2 Proposed Algorithm resource allocation

4.1.2 Experimental and Results

Proposed power and fault aware VM allocation algorithms is simulated using CloudSim 3.0 and power module package. CloudSim provides a benchmark for simulation of cloud platform and also provides Linear power model for simulation of power model. Proposed algorithm is being tested under various request count with 4 servers S1, S2, S3, S4 & S5. Linear power model directly depend on utilization of servers. Proposed algorithm is compared with basic DVFS (Dynamic voltage and frequency scaling) scheduling. Compression of proposed algorithm is performed for 200, 400, 600, 800, 1200 and 1400 set of requests. These set of request contributes of various type of short, average and large requests sizes. System configuration taken into consideration is as follows:

Table 5

Server	RAM(Mb)	MIPS	Storage (Gb)	core	PE	HOST
S1	2000	10000	100000	4	10	2
S2	2000	10000	100000	6	10	2
S3	2000	10000	100000	6	10	2
S4	2000	10000	100000	6	10	2
S5	2000	10000	100000	6	10	2

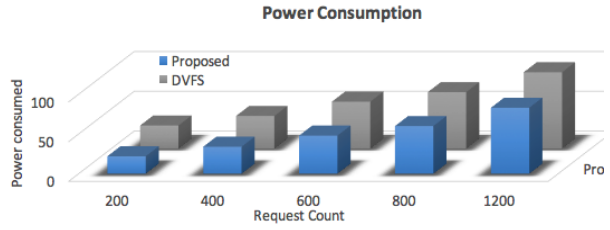


Fig. 4.3 Power consumption

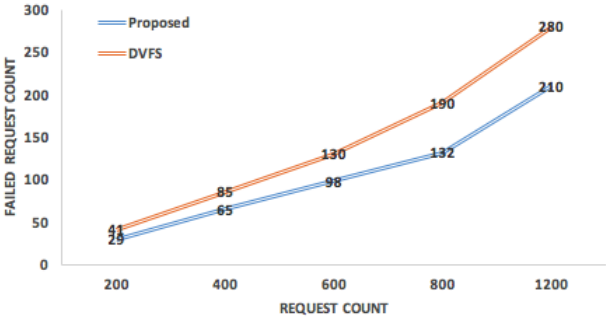


Fig. 4.4 Request Failed

Figure 4.3 shows the improvement in power consumption by proposed algorithm over DVFS. Figure 4.4 shows the improvement in number of request failed by proposed algorithm over DVFS in various test cases.

4.2. Approaches 2: Trust and Deadline-Aware Scheduling Algorithm for Cloud Infrastructure Using Ant Colony Optimization

This paper proposed a trust and deadline aware algorithm that uses various parameters to evaluate the trust value for a host, on that trust value we we have proposed and VM allocation policy to maximize the utilization of resources available in data center. Flow chart of proposed algorithm is below. As can see in flow chart we begin with task pool, here we look for task, if task pool is empty then do nothing but if there is some request in task pool for completion then we proceed to collect the information of each data center .Trust can be defined as an indirect reliability or a firm believe over a host based on its past performance parameters.

Trust is based on:
Start time: Time taken by host to initialize a virtual machine (VM).
Processing Speed: Total number of MIPS of a machine i.e. Number of processor * number of MIPS in each processor
Fault Rate: This can be defined as the total count of request failed over a period of time T
Utilization: this is the current utilization of that host in real time.
Power Efficiency: the ratio of the output power over the input power i.e. the percentage power consumed over a period of time.

For scheduling algorithm, we have proposed an ant colony based VM allocation algorithm which uses a fitness function based on above discussed trust value and deadline to find the fittest host among all.

Trust value (T_i): trust value for host i.

$$T_i = \alpha_1 * Initial_i + \alpha_2 * PS_i + \alpha_3 * \frac{1}{Fault_i} + \alpha_4 * \frac{1}{Utilization_i} + \alpha_5 * \frac{1}{PE_i} \quad (12)$$

$\alpha_1<1$, $\alpha_2<1$, $\alpha_3<1$, $\alpha_4<1$, $\alpha_5<1$ (13)

$$\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 + \alpha_5 = 1 \quad (14)$$

Pseudo code of TDARP (Trust and Deadline Aware Resource Allocation Policy) algorithm takes data centers list, queue length of task in task pool, Power Efficiency of data centers, as shown in pseudo code if task pool is not empty, then calculate the power efficiency on the basis of their utilization.

4.2.1. Proposed algorithm

Algorithm :TDARPA

1. TDARP (Host List PD and Q_{length})
   Input: Host List PD and Queue length Q_{length}
2. PD← Host List
3. i ←No. of Data Centers
4. Q_{length}← current queue size
5. PEi ← Power Efficiency of Host Pd_i
6. Fpi ← Failure Probability of Host Pd_i
7. Initiali ← Start time of Host Pd_i;
8. If(Q_{length} !=0)
9. Sent_Search_ant(Reqtype);
10. find Fittest host(Reqtype);
11. Update pheromone();
12. Q_{length} - - ;
13. If(Q_{length} >0) then goto step 1;
14. End

Output: The server with minimum fitness value.

Fig.4.5. Proposed Algorithm (1)

Algorithm : Find fittest host

1. Find_Fittest host(Reqtype r)
   Input: Request type r
2. Compute PEi for each host
3. Compute utilization for each host
4. Compute Ti for each host
5. Sort_decending(Ti)
6. Find host with high trust value and fit's deadline
7. Return selected host

Output: The server with minimum fitness value.

Fig. 4.6. Proposed algorithm (2)

4.2.2 Experiment and Results

Proposed power based trust and deadline aware allocation algorithms is simulated using cloudSim 3.0 and power module package. Linear power model is used for simulation of power model. Proposed algorithm is being tested under various request count with 4 servers S1, S2, S3, S4. Linear power model directly depend on utilization of servers. Proposed algorithm is compared with basic DVFS (Dynamic voltage and frequency scaling) scheduling. Compression of proposed algorithm is performed for 1000, 1500, 2000, 2500 and 3000 set of requests. System configuration taken into consideration is as follows:

Table 6

Server	RAM (Mb)	MIPS	Storage (Gb)	core	PE	HOST
S1	3000	2000	100000	4	10	4
S2	3000	2000	100000	6	10	4
S3	3000	2000	100000	4	10	4
S4	3000	2000	100000	4	10	4

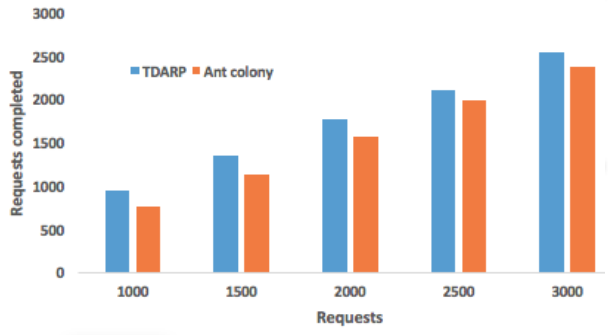


Fig.4.7. Comparison of request completed

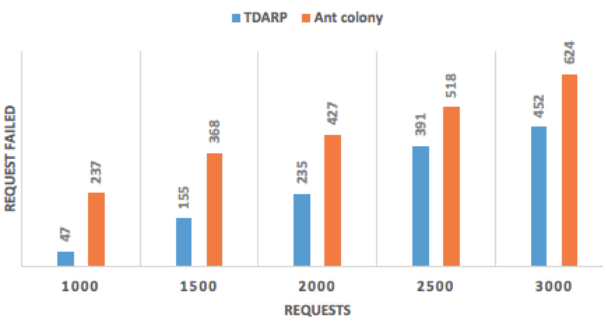


Fig. 4.8. Comparison of request failed

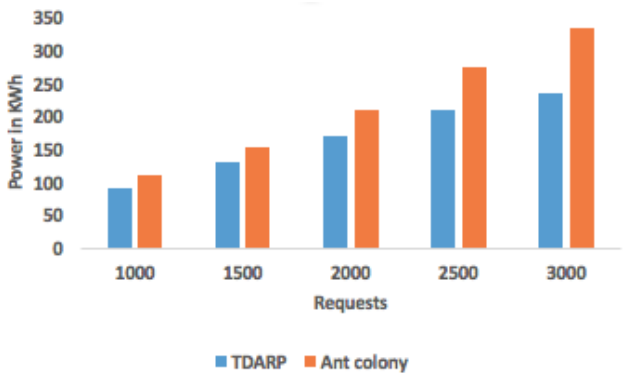


Fig. 4.9. Comparison of power consumed in KWh

4.3. Summary and Discussion

The main achievement of this study and work was to study the rich literature and solve the issue of resource allocation in fault aware cloud environment. The results obtained with our approach were very competitive with most of the well known algorithms in the literature and justified over the large collection of requests. Proposed resource allocation algorithm proves to provide better fault tolerance as compared to existing algorithm with least request failure, high request completion count, and Power consumption of system.

4 CONCLUSION

The work puts its eye to achieve fast, optimal and fault tolerant algorithm for load and resource allocation for cloud. Novel approaches to improve the performance of system in variant load and cloud environment have been presented. The issues of resource optimization with high resource utilization are optimized with proposed load balancing techniques. Proposed algorithm also optimizes the scheduling time to allocate all resource with high resource utilization by promising global best schedule with least failure probability. Finally, as depicted by the experiments, proposed approaches are proven to be superior as compared to existing algorithm. The efforts are made to design algorithms in such a way to improve over system performance rather than improving a specific parameter like cost, deadline or power.

5 REFERENCES

[1]. Hsiao HC, Chung HY, Shen H, Chao YC. Load rebalancing for distributed file systems in clouds. Parallel and Distributed Systems, IEEE Transactions on. 2013 May;24(5):951-62.

[2]. Anuradha VP, Sumathi D. A survey on resource allocation strategies in cloud computing. In Information Communication and Embedded Systems (ICICES), 2014 International Conference on 2014 Feb; 27:1-7.

[3]. Kansal NJ, Chana I. Existing load balancing techniques in cloud computing: A systematic review. Journal of Information Systems and Communication. 2012 Jan; 1;3(1):87.

[4]. Yamamoto H, Maruta D. Replication methods for load balancing on distributed storages in P2P networks. IEICE transactions on information and systems. 2006 Jan; 1;89(1):171-80.

[5]. Rao A, Lakshminarayanan K, Surana S, Karp R, Stoica I. Load balancing in structured P2P systems. In Peer-to-Peer Systems II 2003 Feb; 21 :68-79.

[6]. Zeng W, Li Y, Wu J, Zhong Q, Zhang Q. Load Rebalancing in Large-Scale Distributed File System. In Information Science and Engineering (ICISE), 2009 1st International Conference on 2009 Dec; 26 :265-269.

[7]. Fan K, Zhang D, Li H, Yang Y. An adaptive feedback load balancing algorithm in HDFS. In Intelligent Networking and Collaborative Systems (INCoS), 2013 5th International Conference on 2013 Sep; 9 :23-29.

[8]. Chung HY, Chang CW, Hsiao HC, Chao YC. The load rebalancing problem in distributed file systems. In Cluster Computing (CLUSTER), 2012 IEEE International Conference on 2012 Sep; 24 :117-125.

[9]. Selvarani S, Sadhasivam GS. Improved cost-based algorithm for task scheduling in cloud computing. In Computational intelligence and computing research (iccic), 2010 IEEE international conference on 2010 Dec; 28 :1-5.

[10]. Beloglazov A, Abawajy J, Buyya R. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. Future generation computer systems. 2012 May; 31;28(5):755-68.

[11]. Yao J, He JH. Load balancing strategy of cloud computing based on artificial bee algorithm. In Computing Technology and Information Management (ICCM), 2012 8th International Conference on 2012 Apr; 24:185-189.

[12]. Kwannetr U, Leeton U, Kulworawanichpong T. Optimal power flow using artificial bees algorithm. In Advances in Energy Engineering (ICAEE), 2010 International Conference on 2010 Jun; 19 :215-218.

[13]. Dorigo M, Blum C. Ant colony optimization theory: A survey. Theoretical computer science. 2005 Nov 17;344 (2):243-78.

[14]. Lu Y, Zhang J, Wu S, Zhang S. A Hybrid Dynamic Load Balancing Approach for Cloud Storage. In Industrial Control and Electronics Engineering (ICICEE), 2012 International Conference on 2012 Aug; 23 :1332-1335.

[15]. Zeng Z, Veeravalli B. Design and performance evaluation of queue-and-rate-adjustment dynamic load balancing policies for distributed networks. Computers, IEEE Transactions on. 2006 Nov; 55(11):1410-22.
[16]. Alakeel AM. A guide to dynamic load balancing in distributed computer systems. International Journal of Computer Science and Information Security. 2010 Jun; 10(6):153-60.
[17]. Escalnte D, Korty AJ. Cloud services: policy and assessment. Educause Review. 2011 Jul; 46(4).
[18]. Brown R. Report to congress on server and data center energy efficiency: Public law Lawrence Berkeley National Laboratory 2008 Jun; 13:109-431.
[19]. Hu J, Gu J, Sun G, Zhao T. A scheduling strategy on load balancing of virtual machine resources in cloud computing environment. In Parallel Architectures, Algorithms and Programming (PAAP), 2010 Third International Symposium on 2010 Dec; 18 :89-96.
[20]. Hu J, Gu J, Sun G, Zhao T. A scheduling strategy on load balancing of virtual machine resources in cloud computing environment. In Parallel Architectures, Algorithms and Programming (PAAP), 2010 Third International Symposium on 2010 Dec; 18 :89-96.
[21]. Bohrer P, Elnozahy EN, Keller T, Kistler M, Lefurgy C, McDowell C, Rajamony R. The case for power management in web servers. In Power aware computing 2002 :261-289.
[22]. Fan X, Weber WD, Barroso LA. Power provisioning for a warehouse-sized computer. In ACM SIGARCH Computer Architecture News 2007 Jun 9:53(2):13-23.
[23]. Suraj SR, Natchadalingam R. Adaptive Genetic Algorithm for Efficient Resource Management in Cloud Computing. International Journal of Emerging Technology and Advanced Engineering. 2014 Feb;4 (2):350-6.
[24]. Buyya R, Ranjan R, Calheiros RN. Modeling and simulation of scalable Cloud computing environments and the CloudSim toolkit: Challenges and opportunities. In High Performance Computing & Simulation, 2009. HPCS'09. International Conference on 2009 Jun; 21 :1-11.

6 LIST OF PUBLICATIONS

Journal Publications

[1] Gupta P, Ghrera SP Load Balancing Algorithm for Hybrid Cloud IaaS. International Journal of Applied Engineering Research (IJAER). 2015 10:69 :257-261. ***[SCOPUS, EBSCOhost, GOOGLE Scholar] (SJR: 0.13) (Cite 1)***

[2] Gupta P, Ghrera SP. Fault and Load Aware Load Balancing in Cloud Storage. International Journal of Applied Engineering Research (IJAER). 2015 10:69 :280-285**.** ***[SCOPUS, EBSCOhost, GOOGLE Scholar] (SJR: 0.13) (Cite 1)***

[3] Gupta P, Ghrera SP. Deadline-Aware Load Balancing of Distributed Servers in Distributed. International Journal of Applied Engineering Research (IJAER). 2015 10:69 :268-273 ***[SCOPUS, EBSCOhost, GOOGLE Scholar] (SJR: 0.13)***

[4] Gupta P, Ghrera SP. Fault and QoS based Genetic Algorithm for Task Allocation in Cloud Infrastructure. Indian Journal of Science and Technology (IJST*).* ***[SCOPUS, ISI Thomson Reuters, DOAJ, Copernicus Poland] (SJR: 1.3) (Accepted)***

[5] Gupta P, Ghrera SP. Fault Task Allocation Using Big Bang-Big Crunch in Cloud Infrastructure. Indian Journal of Science and Technology (IJST). ***[SCOPUS, ISI Thomson Reuters, DOAJ, Copernicus Poland] (SJR: 1.3) (Accepted)***

[6] Gupta P, Ghrera SP. Fault Tolerant Big Bang-Big Crunch for Task Allocation in Cloud Infrastructure. International Journal of Advanced Intelligence Paradigms, (Accepted). ***[SCOPUS, ACM Digital Library, DBLP, GOOGLE Scholar] (SJR: 0.16)***

[7] Gupta P, Ghrera SP. QoS Aware Grey Wolf Optimization For Task Allocation In Cloud Infrastructure. . International Journal of Advanced Intelligence Paradigms, (Communicated). ***[SCOPUS, DBLP, GOOGLE Scholar] (SJR: 0.12)***

Conference Proceedings

[8] Gupta P, Ghrera SP. Load and Fault Aware Honey Bee Scheduling Algorithm for Cloud Infrastructure. In Proceedings of the 3rd International Conference on Frontiers of Intelligent Computing: Theory and Applications (FICTA) 2014 2015 :135-143. Springer International Publishing. ***[SCOPUS, ISI Thomson Reuter, DBLP] (Cite 3)***

[9] Gupta P, Ghrera SP. Power and Fault Aware Reliable Resource Allocation for Cloud Infrastructure. Procedia Computer Science. 2016 Dec 31;78:457-63. ***[SCOPUS, Proceedings Citation Index, GOOGLE Scholar] (Impact: 0.705)***

[10] Gupta P, Ghrera SP. Trust and Deadline-Aware Scheduling Algorithm for Cloud Infrastructure Using Ant Colony Optimization2016 International Conference on Innovation and Challenges in Cyber Security (ICICCS) 2016 Feb. ***[SCOPUS]***